

Trabalho prático 2

Redes de Computadores

Rede de Computadores

Trabalho realizado por:

Nuno Oliveira [up201806525@fe.up.pt](mailto:up201806525@fe.up.pt)

Luís Pinto [up201806206@fe.up.pt](mailto:up201806206@fe.up.pt)

**Sumário**

Este relatório descreve a arquitetura da aplicação de download que implementa um cliente simples do protocolo FTP, como descrito no RFC959, por nós desenvolvida bem como os resultados da mesma. Para além disso, é feita uma análise sobre a série de experiências relativas à segunda parte deste segundo trabalho prático.

**Introdução**

Para o segundo trabalho prático de Redes de Computadores foi-nos proposto o desenvolvimento de uma aplicação que faz download de um único ficheiro implementado um cliente FTP bem como a configuração de uma rede e análise da mesma. Os objetivos deste trabalho são entender o comportamento do protocolo FTP, usar sockets e TCP na linguagem C, perceber DNS do serviço e usá-lo com o programa cliente e, por fim, entender a arquitetura de uma rede de computadores.

**Aplicação de Download**

A aplicação deve receber como argumento da consola um link do tipo: **ftp://[<user>:<password>@]<host>/<url-path>** de acordo com a sintaxe descrita no RFC1738. Com este link devemos tentar conectar ao host indicado com as credenciais fornecias para fazer download do ficheiro indicado no url-path.

**Arquitetura**

O primeiro passo é então fazer parse do link recebido como argumento e para tal usamos a função **getArgsFromUrl** que recebe como argumentos o link e a struct a preencher com os valores lidos em caso de não haver erros.

De seguida recorremos à função **getIP** que, com o nome do host, obtém o IP do mesmo para passar à função **openSocket** que, por sua vez, cria e conecta um socket Berkely na porta 21.

Os próximos passos envolvem enviar e receber comandos do servidor e para tal usamos as funções **writeToSocket** e **readCommandFromSocket**.

Usando então estas duas funções, depois de aberto o socket, é necessário fazer login usando os comandos “user example-user” e “pass exemple-pass”. Se o ficheiro não estiver no root do servidor é também necessário mudar o diretório atual usando o comando “cwd diretório”.

De seguida é necessário entrar no modo passivo com o comando “pasv” e ler, da resposta, o IP e a porta do qual será lido o conteúdo do ficheiro. Para ler a resposta e obter o IP é usada a função **getIPFromBody**, que guarda o IP e a porta das duas variáveis passados como argumentos. A seguir é criada outra socket que se conecta ao novo IP e porta.

Finalmente, enviando o comando “retr nome-do-ficheiro” para o primeiro socket aberto, podemos ler do socket mais recente o conteúdo do ficheiro.

**Resultados de um download com sucesso**

Se não houver erros no link passado à aplicação nem na conexão ao servidor pode-se observar no terminal outputs parecido com o da figura 1 nos anexos. Depois de alguns testes concluiu-se que a aplicação funciona como esperado, fazendo download de ficheiros de diferentes tipos e tamanhos, estando ou não no root do servidor.

**Configuração e análise da rede**

Partindo do princípio de que o switch tem as configurações default não há ligações entre nenhum dos aparelhos:

**Experiência 1 – Configurar uma rede IP**

O objetivo desta experiência foi ligar os computadores 3 e 4 ao switch, configurar a interface de rede eth0 de ambos com o IP e máscara corretos para que fosse possível enviar mensagens entre os dois computadores.

Principais comandos:

* “ifconfig <interface de rede> <IP>/<máscara>” - atribuição de IPs a uma interface de rede

Questões sobre esta experiência:

* O que são os pacotes ARP e para que são usados?

ARP (Address Resolution Protocol) é um protocolo definido pela RFC 826 e tem como função mapear o endereço físico de uma máquina (MAC) na rede ao seu endereço lógico (IP). Para tal primeiro é feito um pedido em *broadcast* (à rede toda), enviando um pacote ARP a perguntar o endereço MAC da máquina com o endereço de IP pretendido. A máquina com o endereço lógico indicado irá enviar um pacote ARP de resposta em *unicast* (com apenas um destino) para a máquina que fez inicialmente o pedido com o seu endereço MAC. No fim, ambos guardam os dados um do outro em cache (tabela ARP).

* Quais são os endereços MAC e IP dos pacotes ARP e porquê?

Como quando o gnu3 tenta enviar um pacote ao endereço lógico do gnu4, não sabe qual o endereço físico deste último, envia então um pacote ARP em *broadcast* com o seu endereço de IP e MAC.

O pacote ARP de pedido contém o IP (172.16.10.1) e o endereço MAC (00:21:5a:5a:7d:16) como *source* para que quem responder ao pedido poder também guardar estes valores em cache e, efetivamente, responder. Como *destination,* contém o IP (172.16.10.254) e MAC (00:00:00:00:00:00) sendo que se sabe obviamente qual o endereço lógico da máquina de destino (indicado no comando *ping*), mas não o físico.

O pacote ARP de resposta vem do gnu4 e contém como *destination* os endereços que o pacote de pedido tinha como source e, por sua vez, como *source* tem o endereço de IP (172.16.10.254) e MAC (00:21:5a:5a:7d:3f) indicando então o seu endereço físico ao gnu3.

* Que pacotes gera o comando *ping*?

Se o endereço indicado no comando não estiver guardado na tabela ARP, gera primeiro um pacote ARP de *request*. Depois disso gera e envia pacotes ICMP (Internet Control Message Protocol).

* Quais são os endereços MAC e IP dos pacotes gerados pelo comando *ping*?

Os pacotes gerados no gnu3 e recebidos no gnu4 (pacotes *request*) têm como *destination* os endereços IP (172.16.10.254) e MAC (00:21:5a:5a:7d:3f) e como *source* os endereços IP (172.16.10.1) e MAC (00:21:5a:5a:7d:16).

Os pacotes gerados no gnu4 e recebidos no gnu3 (pacotes *reply*) têm como *destination* os endereços IP (172.16.10.1) e MAC (00:21:5a:5a:7d:16) e como *source* os endereços IP (172.16.10.254) e MAC (00:21:5a:5a:7d:3f).

* Como determinar se uma *frame* Ethernet é ARP, IP ou ICMP?

O tipo de identifica-se verificando, no *header* da mesma, os bytes 13 e 14 da *frame*. Se tiver o valor 0x0806 é um pacote ARP e tiver o valor 0x0800 é um pacote IP. Considerando que o protocolo ICMP é integrante do protocolo IP estes diferenciam-se pelo byte 24, se tiver o valor 0x01 é ICMP. Os bytes foram contados começando por 1 e não por 0 ao analisar os pacotes capturados pelo wireshark.

* Como determinar o tamanho de uma *frame* recebida?

O comprimento total da trama pode ser visualizado recorrendo ao software wireshark. Pode-se ver diretamente o tamanho duma *frame* mas, analisando melhor reparamos que os 12 primeiros bytes correspondem aos endereços MAC do destino e fonte do pacote, por ordem, os próximos 2 indicam o tipo de pacote, e o restante é conteúdo do protocolo.

* O que é a interface *loopback* e porque é importante?

A interface *loopback* é uma interface de rede virtual que permite a um dispositivo de comunicação enviar pacotes destinados a si mesmo para identificar problemas na rede. Caso haja algum problema de comunicação, utiliza-se *loopback* para entender se o problema está no outro dispositivo (se receber a *frame* enviada a si mesmo) ou na rede (se não receber a *frame* enviada a si mesmo).

**Experiência 2 – Implementar duas virtual LANs num switch**

O objetivo desta experiência foi criar duas LANs virtuais, vlan10 e vlan11, adicionar os gnu3 e gnu4 à vlan10, o gnu2 à vlan11 e testar a comunicação entre eles todos.

Comandos principais:

* “ifconfig” – usado novamente para configurar a interface eth0 do gnu2
* Todos os comandos indicados na resposta à primeira pergunta desta experiência que indicam com criar uma vlan.

Questões sobre esta experiência:

* Como configurar uma vlan (no nosso caso vlan10 e vlan11)?

Para criar uma vlan é apenas necessário introduzir os seguintes comandos no GTKTerm do computador ligado à consola do switch:

* + configure terminal
  + vlan 10
  + end

Repetir os passos para a vlan 11.

Para adicionar um computador a determinada vlan é primeiro necessário que este tenha a porta eth0 ligada ao switch e que esteja configurada. Depois introduz-se os seguintes comandos no GTKTerm:

* + configure terminal
  + interface fastethernet 0/y
  + switchport mode access
  + switchport access vlan xx
  + end

Substituir “y” pelo número da porta no switch em que está ligada a porta eth0 do computador em questão e substituir “xx” pela vlan à qual se pretende adicionar o computador.

* Quantos domínios de *broadcast* existem e como deduzir isso dos *logs* obtidos?

Existem 2 domínios de *broadcast*, um com o gnu3 e o gnu4 e outro apenas com o gnu2. Podemos concluir isso a partir dos *logs* porque o *ping* enviado em *broadcast* a partir do gnu3 chega ao gnu4, mas não ao gnu2.

**Experiência 3 – Configurar um router em Linux**

Nesta experiência o gnu4 foi configurando como um router para estabelecer a ligação entre as VLANs 10 e 11

Principais comandos:

* echo 1 > /proc/sys/net/ipv4/ip\_forward – ativar ip forwarding
* echo 0 > /proc/sys/net/upv4/icmp\_echo\_ignore\_broadcasts – desativar ICMP echo-ignore-broadcast
* route add -net <network> gw <IP> - manualmente adicionar uma rota

Questões sobre esta experiência:

* Que rotas existem nos computadores e o que significam?

Adicionar um computador a uma vlan gera rotas automaticamente:

- O gnu3 tem rota para a vlan10 pela gateway 172.16.10.1

- O gnu4 tem rotas para a vlan10 pela gateway 172.16.10.254 e para a vlan11 pela gateway 172.16.11.253

- O gnu2 tem rota para a vlan11 pela gateway 172.16.11.1

Para além destas há mais duas introduzidas manualmente:

- O gnu3 tem rota para a vlan11 pela gateway 172.16.10.254

- O gnu2 tem rota para a vlan 10 pela gateway 172.16.11.253

Isto significa que quando o gnu3 tentar pingar o gnu2, vai primeiro enviar os pacotes ICMP para o gnu4 que, por sua vez, os envia para o gnu2. Isto é possível graças ao ip forwarding ativado no gnu4 (o computador que recebe os pacotes entende que os pacotes não são destinados a ele e então envia para o destino devido).

* Que informações contém uma entrada na tabela de forwarding?

Destination – destino da rota

Gateway – o IP usado para chegar à rota

NetMask – usado para determiner o ID da rede a partir do IP do destino

Flags – informações sobre a rota

Metric – custo da rota

Ref – número de referências para a rota (não usado no kernel do Linux)

Use – contador de pesquisas na rota, dependendo do uso de -F (falhas) ou -C (sucesso)

Interface – placa de rede responsável pelo gateway

* Que mensagens ARP e associados endereços MAC são observados e porquê?

O gnu3, ao tentar pingar o gnu2, irá mandar os pacotes ICMP para o IP indicado na rota adicionada manualmente. No entanto, como as tabelas ARP foram limpas em todos os computadores, o gnu3 não tem o endereço MAC da porta do gnu4 na sua vlan nem o gnu2 tem o endereço MAC da porta do gnu4 na sua vlan. Assim, são gerados pacotes ARP pelos computadores 3 e 2 para guardar os endereços físicos aos quais têm de enviar os pacotes de ICMP*.*

* Que pacotes ICMP são observados e porquê?

Os pacotes ICMP observados são de *request*  e *reply* dado que a comunicação entre os dois computadores em VLANs diferentes foi estabelecida com sucesso. Caso contrário, seriam do tipo *host unreachable*.

* Quais são os endereços IP e MAC associados aos pacotes ICMP e porquê?

Os pacotes ICMP de request são enviados contendo na *source* os endereços MAC e IP do gnu3. Como *destination* têm o endereço IP do gnu2, mas o endereço MAC do gnu4. Isto acontece devido à rota adicionada que indica que, se o gnu3 quiser mandar uma mensagem para o gnu2, tem de a mandar primeiro para o gnu4. O gnu4 ao receber um pacote com um endereço IP diferente do seu e tendo ip forwarding ativado, altera os endereços MAC do pacote ICMP e volta a enviar o pacote, desta vez, para o IP de destino. Os novos pacotes terão como *source* o endereço MAC da porta do gnu4 ligado à vlan11 e como *destination* o endereço MAC do gnu2. Os pacotes ICMP de *reply* passam por um processo semelhante só que no sentido inverso.

**Experiência 4 – Configurar um router comercial e implementar NAT**

Esta experiência tem como objetivo configurar um router comercial, ligando-o à rede do laboratório e à vlan11. Para além disso o router foi configurado para implementar a técnica NAT com o fim de possibilitar a conexão à internet.

Principais comandos:

* Configuração do router com NAT (páginas 45 e 46 do guião do trabalho)

Questões sobre a experiência:

* Como configurar uma rota estática num router comercial?

Na consola do router inserir os seguintes comandos:

- conf t

- ip route [destination] [mask] [gateway]

- exit

* Que caminhos são seguidos pelos pacotes nas experiências?

Com os primeiros logs tirados no passo 4 e o comando traceroute desta experiência podemos verificar que os pacotes enviados do gnu2 com destino ao gnu3 são primeiro enviados ao router comercial. Isto acontece porque a rota para o gnu3 foi removida no gnu2 e então o gnu2 envia os pacotes ao seu router default, neste caso, o comercial, que por sua vez, tendo a rota para o gnu3 (adicionada manualmente) envia os pacotes para a porta do gnu4 na mesma rede. Como antes de se mandar os pings se desativou os *redirects* no gnu2, todos os pings após o primeiro vão seguir o mesmo caminho. No entanto, se observarmos os logs onde voltamos a ativar os *redirects* no gnu2 confirma-se que apenas o primeiro pacote passa pelo router, depois o gnu2 recebe um *redirect* do router, guarda a rota para o gnu3 e os novos pings são enviados diretamente para o gnu4.

* Como configurar NAT num router comercial?

Seguindo o guião na página 46 do guião podemos configurar um router com NAT. De forma resumida, é necessário indicar que endereço de IP se encontra ligado à rede local e qual se encontra ligado à exterior, configurar o NAT overload (permitir o uso concorrente por vários hosts), permitir acesso às redes já criadas e criar as rotas default e para a rede na qual o router não está.

* O que faz NAT?

O principal objetivo do protocolo NAT é modificar os pacotes que passam pelo router, mais especificamente os endereços IP. Esta modificação é feita para preservar a privacidade das máquinas na rede privada local quando tentam comunicar com outras máquinas fora da sua rede. Para além disso permite a comunicação com a internet porque sem NAT um computador que recebesse um pedido qualquer com o IP de um computador duma rede privada não sabia para onde enviar a resposta visto que a rota não estaria definida. Ao modificar os IPs do pedido, o NAT escreve também um número correspondente ao *hash* que representa o computador que fez o pedido para que, ao receber a resposta, o router saiba para onde a redirecionar. Assim, em teoria, para um computador fora da rede privada, todos os computadores dessa mesma rede são representados pelo IP do router.

**Experiência 5 – DNS**

O objetivo desta experiência foi configurar um serviço DNS nos computadores da rede.

Questões sobre esta experiência:

* Como configurar o serviço DNS num host?

Para configurar o serviço DNS é preciso modificar o ficheiro /etc/resolv.conf escrevendo os seguintes comamdos:

- search netlab.fe.up.pt (nome do servidor DNS)

- nameserver 172.16.2.1 (endereço IP do servidor)

* Que pacotes são trocados por DNS e que informação é transportada?

O *host* envia um pedido ao servidor DNS com o nome de um outro *host* e, por sua vez, o servidor responde com um pacote que indica qual o endereço de IP desse *host*.

**Experiência 6 – Conexões TCP**

O objetivo desta experiência foi usar a aplicação desenvolvida na primeira parte deste trabalho para entender o funcionamento do protocolo TCP.

* Quantas conexões TCP são abertas pela aplicação?

São abertas duas ligações TCP. Uma para enviar e receber comandos e outra para receber os dados correspondentes ao ficheiro que se pretende descarregar.

* Em que conexão é transportada a informação FTP de controlo?

Na conexão responsável pelo envio e receção de dados (a primeira criada).

* Quais são as fazes de uma conexão TCP?

Na conexão TCP há 3 fases: estabelecimento da conexão, troca de dados e encerramento da ligação.

* Como funciona o mecanismo ARQ TCP? Quais são os campos TCP relevantes? Que informação relevante se pode observar nos logs?

O mecanismo ARQ TCP utiliza o método janela deslizante. Este método consiste na utilização de *acknowledgment numbers* (indicam se a mensagem foi recebida com sucesso ou não), *window size* (gama de pacotes que podem ser enviados) e *sequence number* (número do pacote que está a ser enviado). Estes dados podem ser observados todos nos logs nos pedidos TCP.

* Como funciona o mecanismo de controlo de congestionamento do TCP? Quais são os campos TCP relevantes? Como mudou a taxa de transferência ao longo do tempo? Está de acordo com o protocolo de controlo de congestionamento?

Para cada conexão é guardada uma janela de congestão que regula o tamanho da janela deslizante limitando assim o número de pacotes em transição. É utilizado o método “slow start” para evitar enviar mais pacotes do que a rede é capaz de enviar. Este método implica começar com uma janela de congestão de tamanho reduzido e aumentá-la (essencialmente duplicando o tamanho da janela) a cada ACK recebido até que haja um *timeout* (indicando congestão da rede) e a esse ponto, o tamanho da janela de congestão é então reduzido de forma a estabilizar.

A taxa de transferência aumentou rapidamente até pouco antes do segundo segundo e estabilizou até pouco depois do terceiro segundo, momento em que se iniciou outra transferência no gnu2, respeitando assim o protocolo de controlo de congestionamento.

* A taxa de transferência é afetada pela aparição de outra conexão TCP? Como?

Como se pode ver no gráfico gerado pelo *wireshark*, aproximadamente a meio do terceiro segundo, que foi quando se começo a transferência no gnu2, o *throughput* no gnu3 começa a diminuir. O download no gnu3 terminou antes de se poder obter melhores resultados, no entanto, se tivesse continuado, o *throughput* continuaria a descer no gnu3 e a aumentar no gnu2 até estabilizar nos dois computadores.

**Anexos**

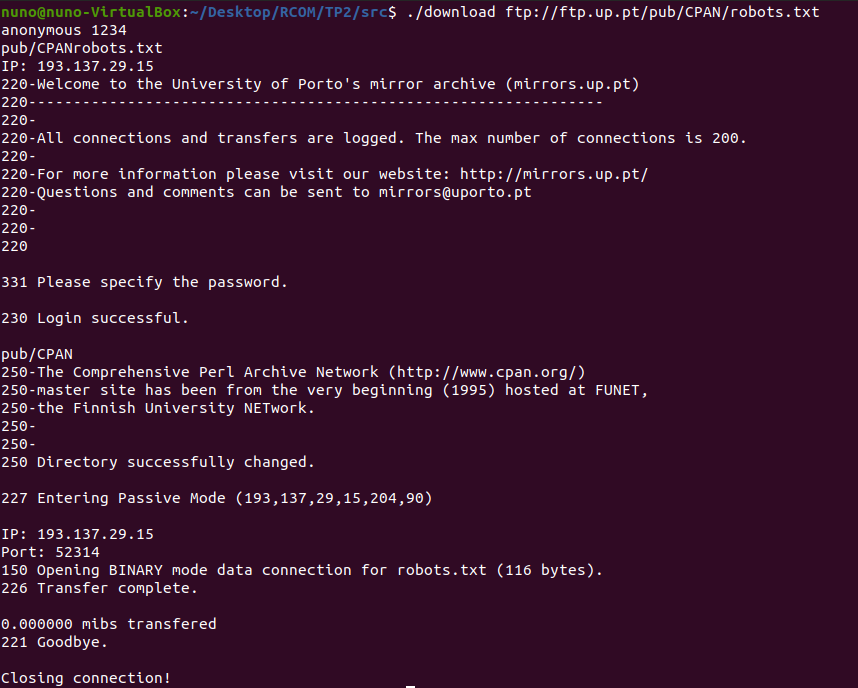


Figura 1 - output da aplicação de download

**Anexos relativos às experiências:**

**Experiência 1**

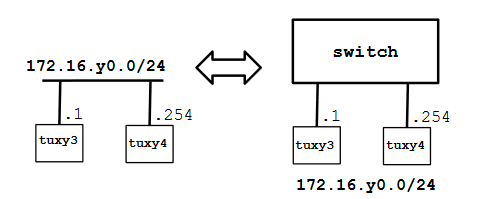


Figura 2 - Topologia da rede

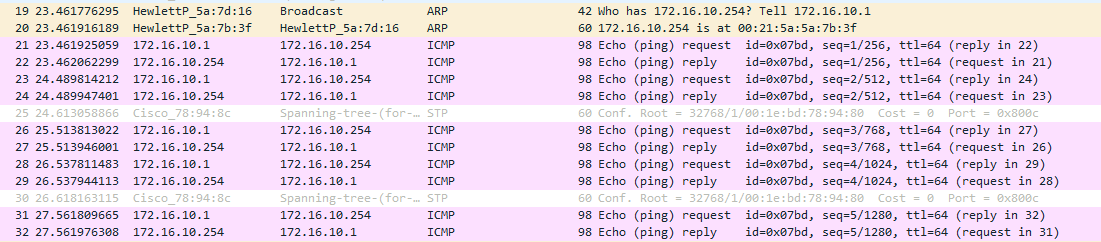


Figura 3 - Mensagens entre gnu3 e gnu4

**Experiência 2**

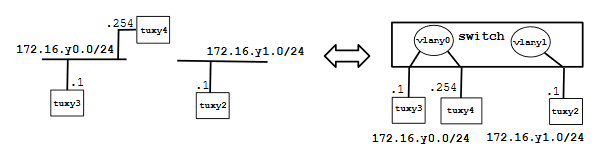


Figura 4 - Topologia da rede

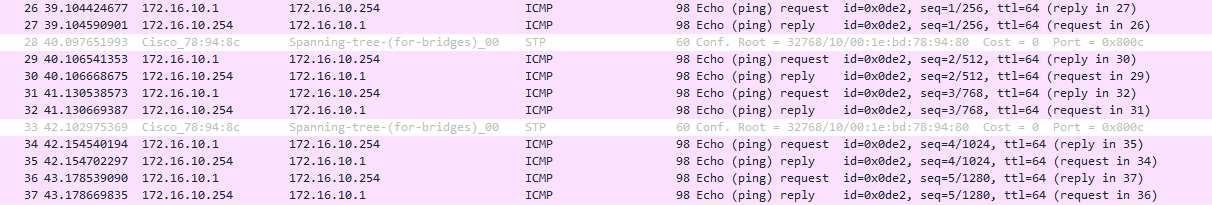


Figura 5 - gnu3 ping gnu4

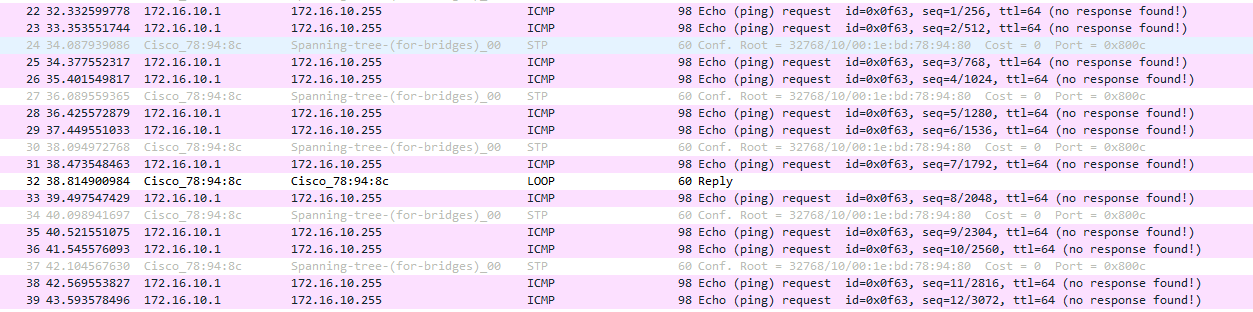


Figura 6 - gnu3 ping broadcast

**Experiência 3**

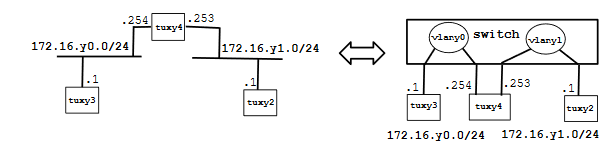


Figura 7 - Topologia da rede

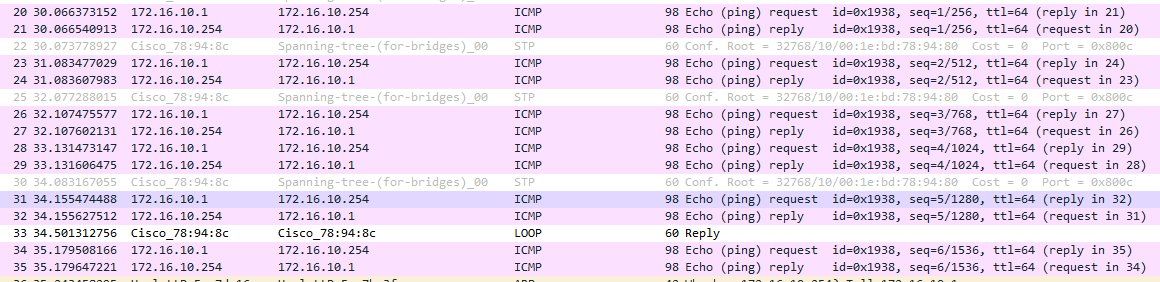


Figura 8 - gnu3 ping gnu4 eth0

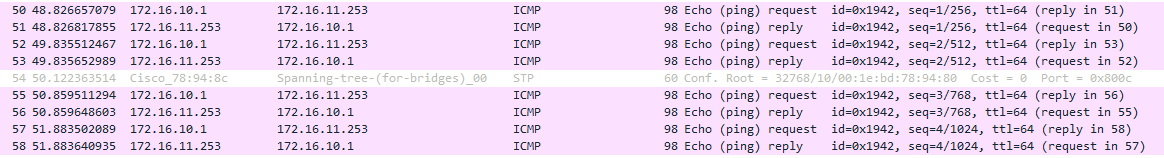


Figura 9 - gnu3 ping gnu4 eth1

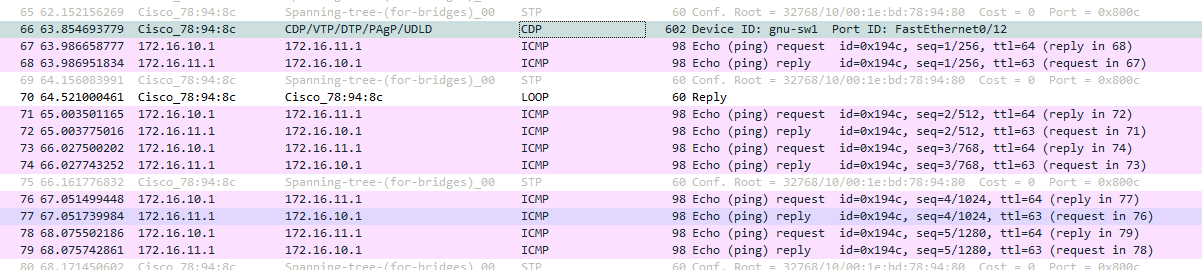


Figura 10 - gnu3 ping gnu2

**Experiência 4**

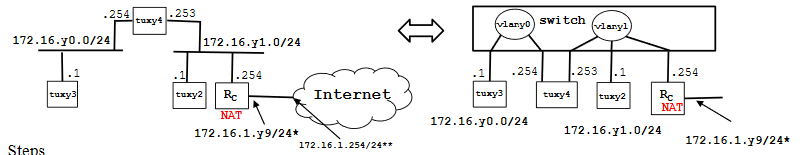


Figura 11 - Topologia da rede

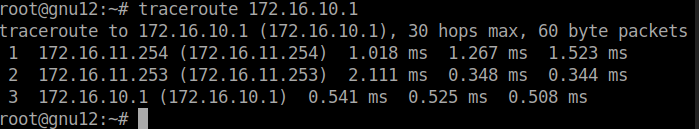


Figura 12 - traceroute do gnu2 para o 3 com redirects desligados

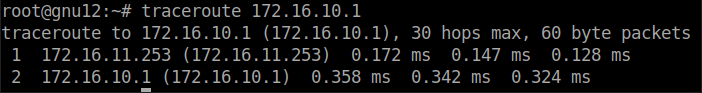


Figura 13 - traceroute do gnu2 para o 3 com redirects ligados

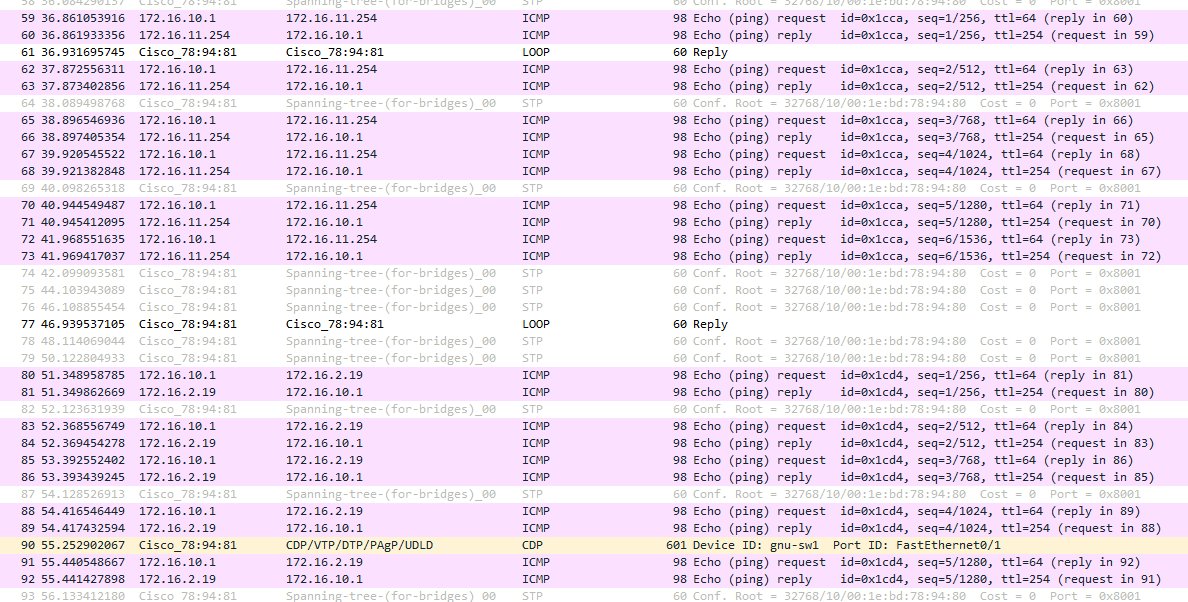


Figura 14 - gnu3 ping router

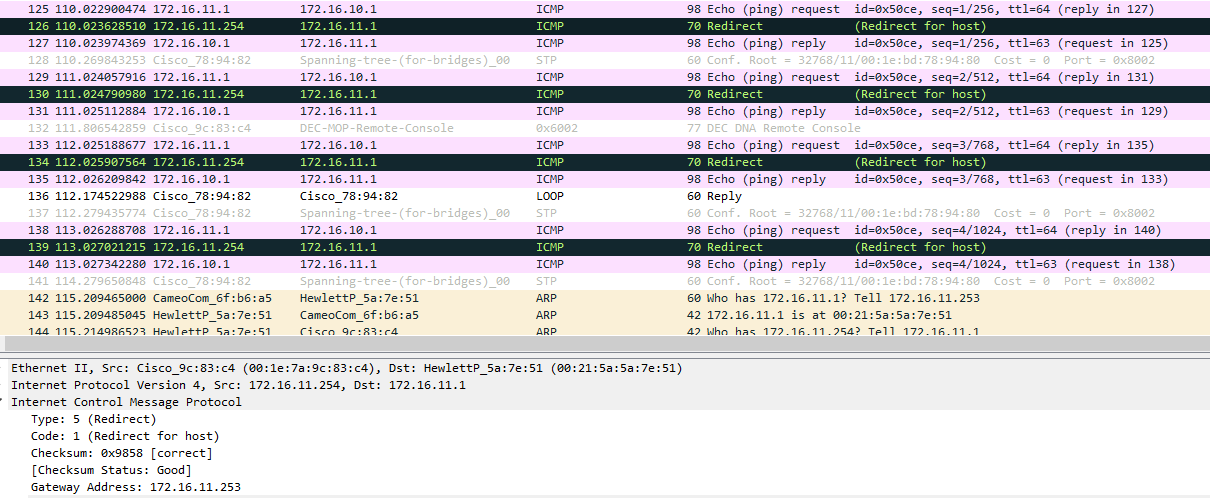


Figura 15 - gnu2 ping gnu3 redirects desligados (passa sempre pelo router antes de ir para o gnu4)

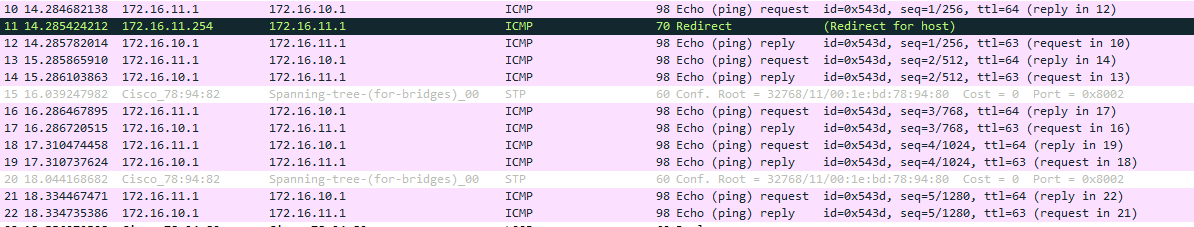


Figura 16 - gnu2 ping gnu3 com redirects ligados (só passa pelo router na primeira vez)

**Experiência 5**

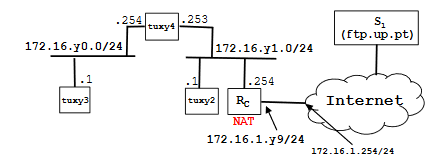


Figura 17 - Topologia da rede

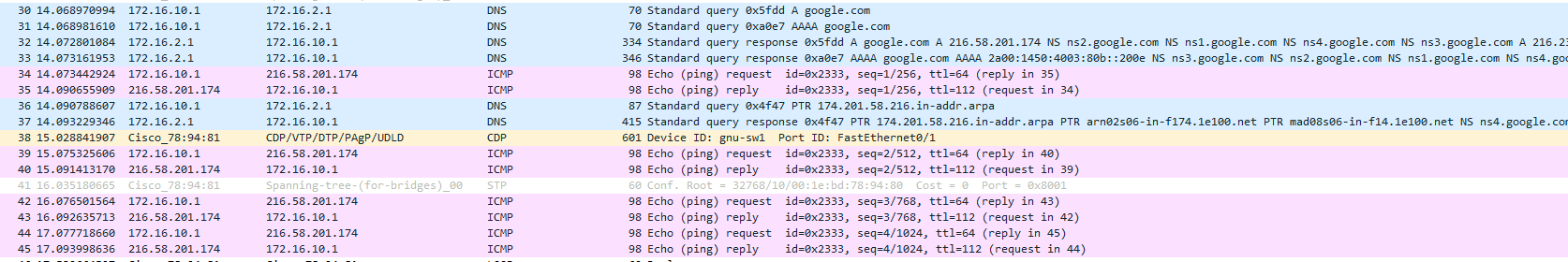


Figura 18 - pacotes DNS ao pingar google.com

**Experiência 6**

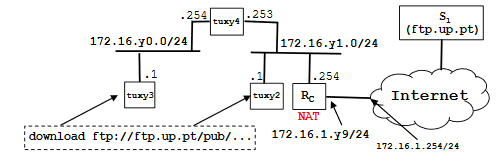


Figura 19 - Topologia da rede

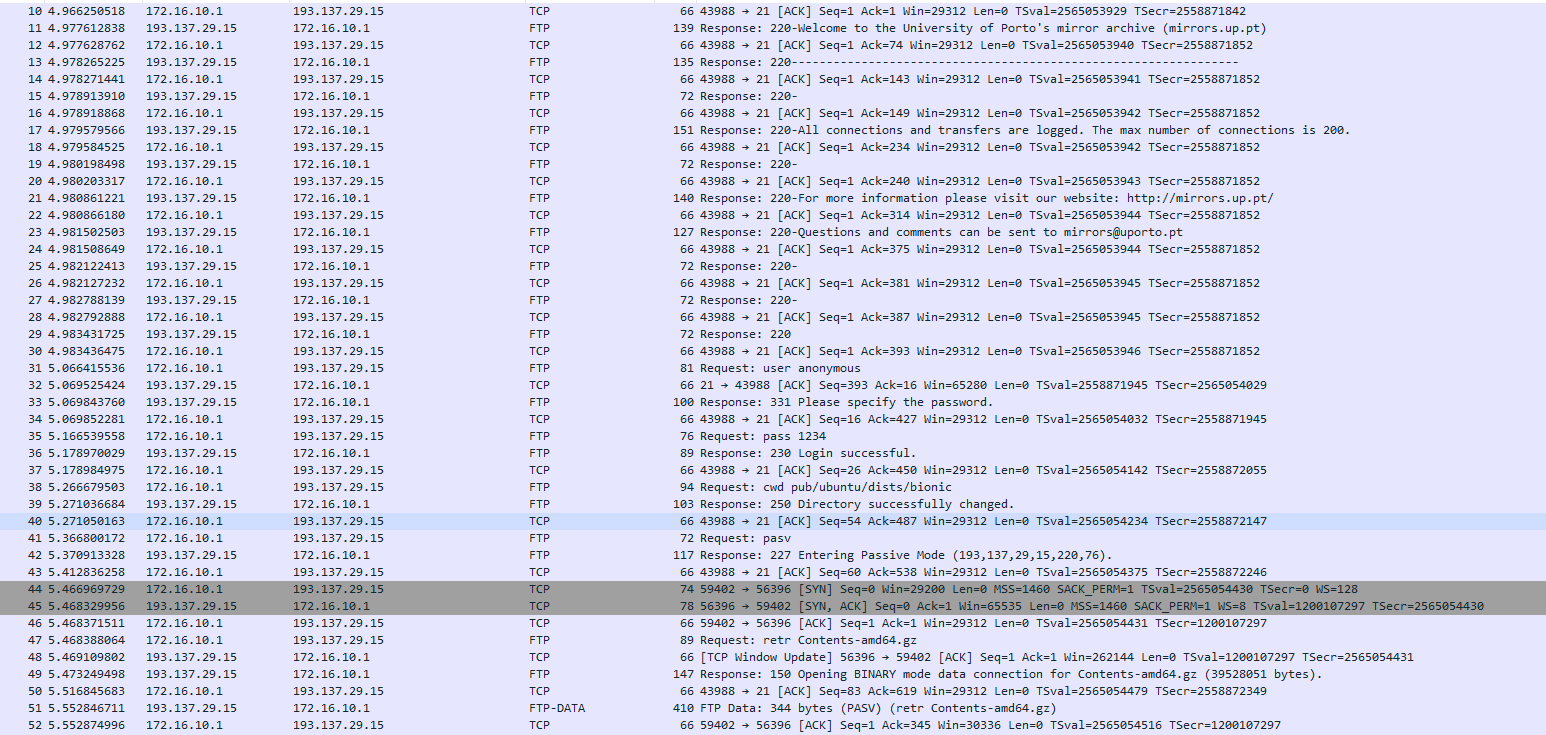


Figura 20 - início da transferência

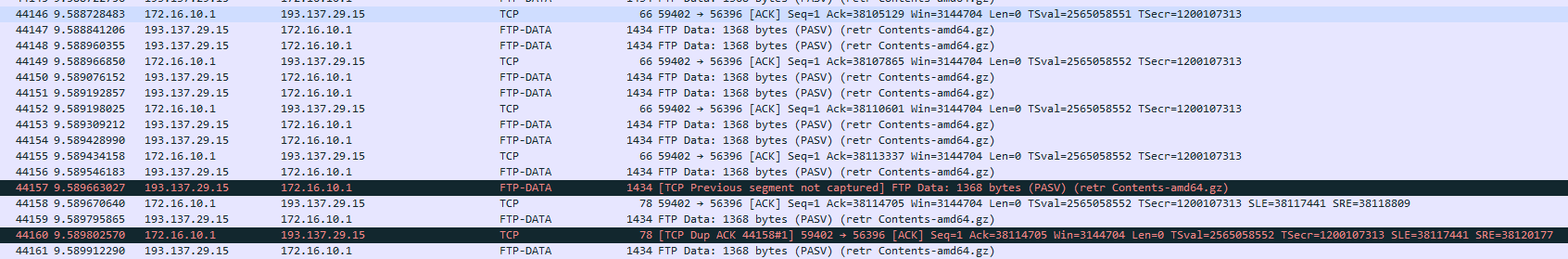


Figura 21 - a meio da transferencia

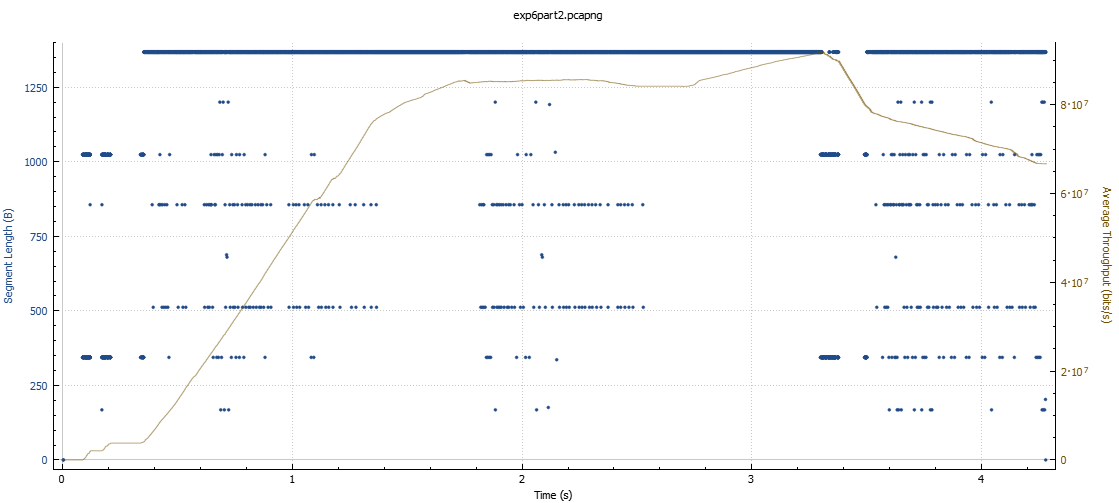


Figura 22 - gráfico de análise de throughput gerado pelo wireshark

**Anexos relativos ao código:**

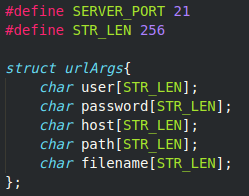


Figura 23 - defines usados ao longo do código e struct que vai conter a informação lida do url passado como argumento

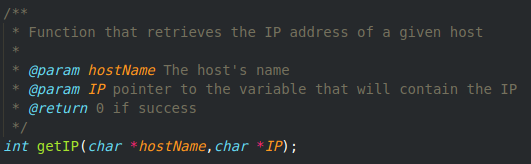


Figura 24 - declaração da função getIP()

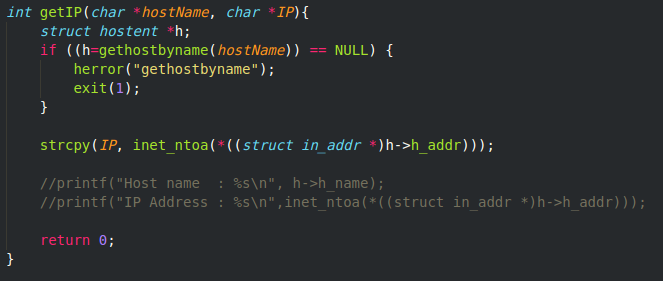


Figura 25 - definição da função getIP()



Figura 26 - declaração da função openSocket()

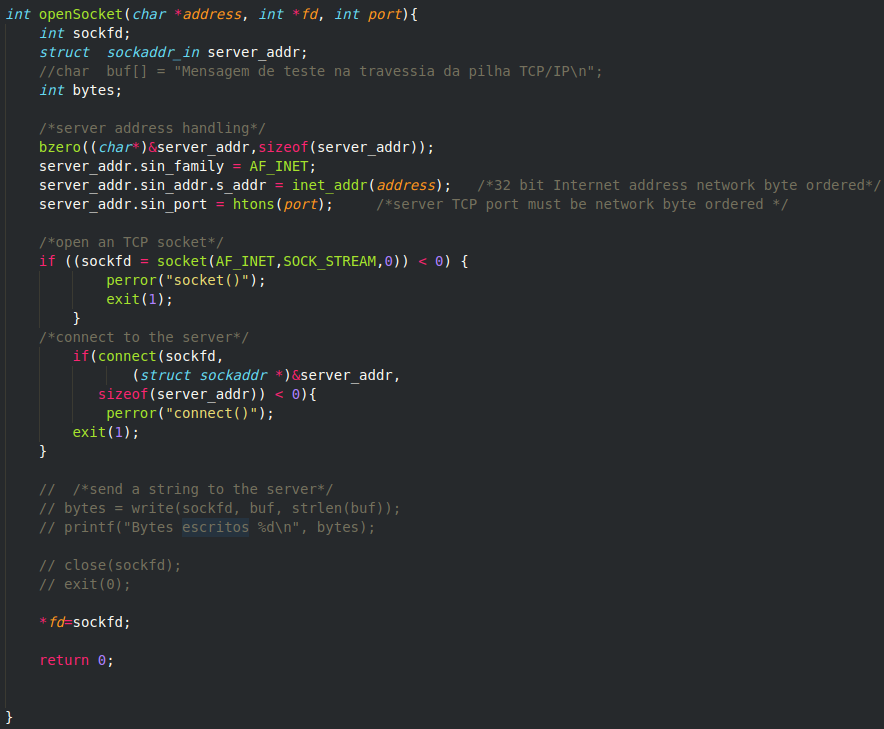


Figura 27 - definição da função openSocket()

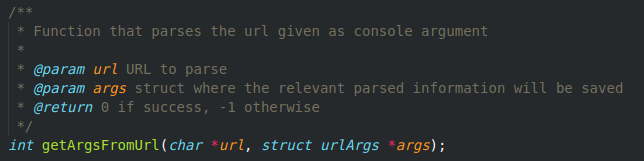


Figura 28 - declaração da função get ArgsFromUrl()



Figura 29 - definição da função getArgsFromUrl() - pt.1

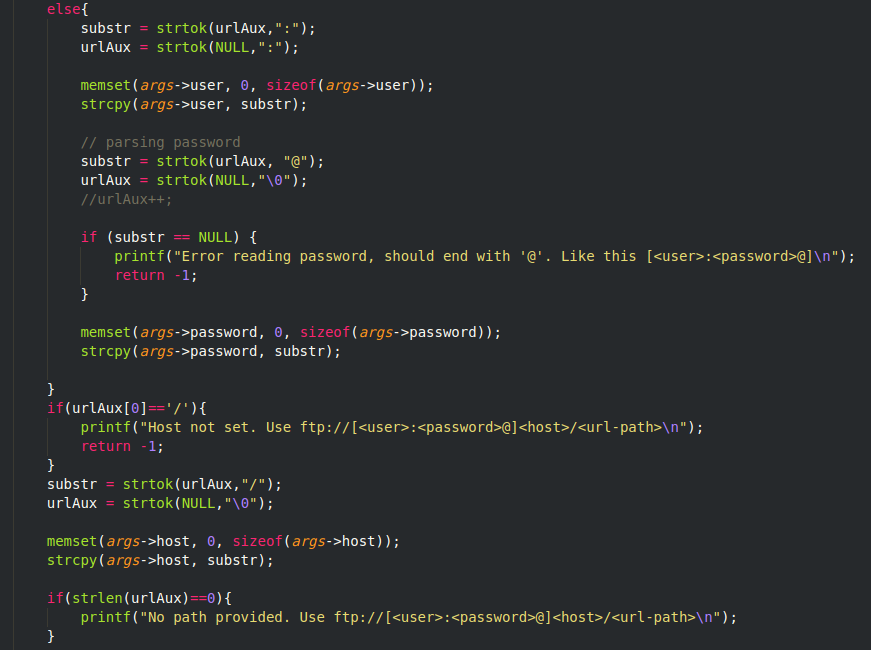


Figura 30 - definição da função getArgsFromUrl() - pt.2

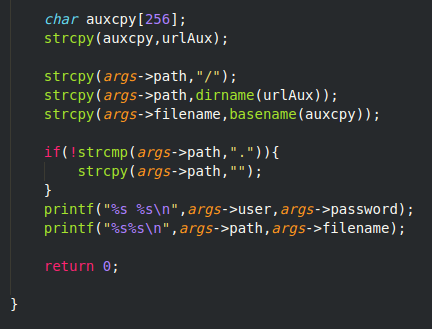


Figura 31 - definição da função getArgsFromUrl() - pt.3

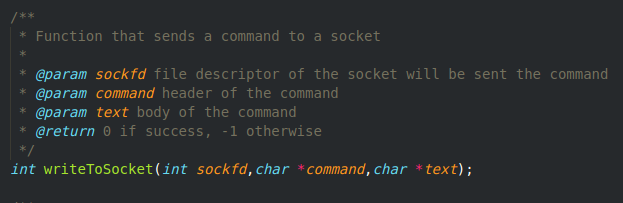


Figura 32 - declaração da função writeToSocket()

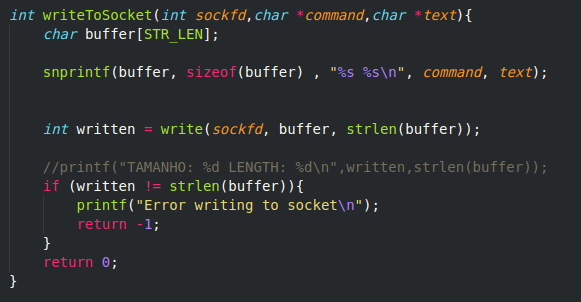


Figura 33 - definição da função writeToSocket()

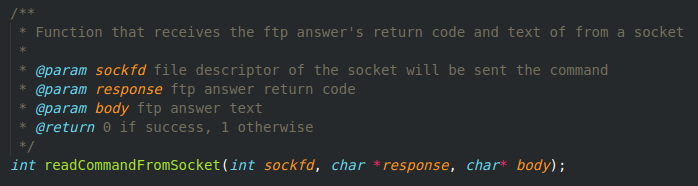


Figura 34 - declaração da função readCommandFromSocket()



Figura 35 - definição da função readCommandFromSocket()

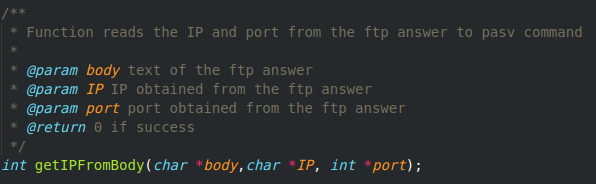


Figura 36 - declaração da função getIPFromBody()

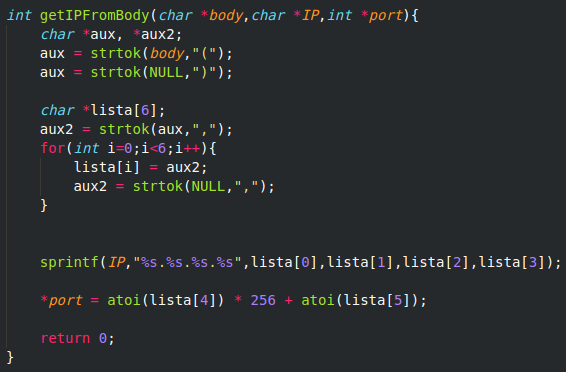


Figura 37 - definição da função getIPFromBody()

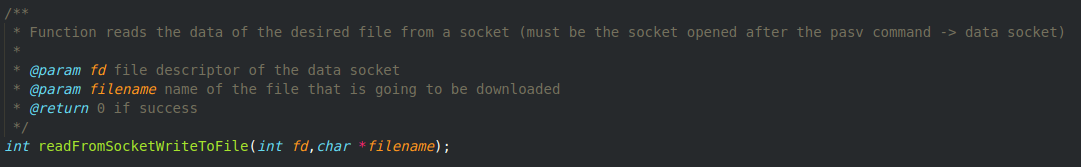


Figura 38 - declaração da função readFromSocketWriteToFile()



Figura 39 - definição da função readFromSocketWriteToFile()



Figura 40 - função main - parse do url, abertura da primeira socket e estabelecimento da conexão



Figura 41 - função main - login e mudança de diretório se for necessário



Figura 42 - envio de pasv, leitura da resposta, abertura da nova socket, receção do ficheiro e término da ligação